15448-0506 *Patent*

UNITED STATES PATENT APPLICATION

FOR

MECHANISM FOR RECOGNIZING AND ABSTRACTING PRE-CHARGED LATCHES AND FLIP-FLOPS

INVENTOR(S):

ALOK JAIN
MANPREET REEHAL

PREPARED BY:
HICKMAN PALERMO TRUONG & BECKER, LLP
1600 WILLOW STREET
SAN JOSE, CALIFORNIA 95125-5106
(408) 414-1080

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number EL652871993US

Date of Deposit December 11, 2001

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Box Patent Application, Commissioner of Patents, Washington, D.C. 20231.

Tirena Say
(Typed or printed name of person mailing paper or fee)

Tirena Say
(Signature of person mailing paper or fee)

MECHANISM FOR RECOGNIZING AND ABSTRACTING PRE-CHARGED LATCHES AND FLIP-FLOPS

Inventor(s): Alok Jain, Manpreet Reehal

Field of the Invention

[0001] This invention relates generally to circuit analysis and more particularly to a mechanism for recognizing and abstracting pre-charged latches and flip-flops.

Background

[0002] In the field of electronic component design, engineers use simulation tools on a regular basis to test their designs. With such tools, engineers are able to detect and correct errors, and enhance the functionality of their designs without actually building a physical component. This leads to great savings in manufacturing and design costs. Over the years, a number of techniques and tools have been developed to aid in the simulation process.

[0003] Currently, simulation tools can simulate and test the behavior of a component design on several different levels. One level at which designs can be simulated is the transistor level. To test a design at this level, an engineer typically specifies a design using a transistor level description language, such as SPICE or SPECTRE. The design is then run through a transistor level simulation tool. Based upon the results of the simulation, the engineer can determine whether the design operated properly. While simulating a design at the transistor level is effective, it is often not the most efficient way to test a design. This is because transistor level simulation is

relatively slow and quite resource intensive, and because designers often are not interested in the operation of a design at that low a level. As a result, it is often preferable to simulate the operation of a design at a higher logic level rather than at the transistor level.

[0004] To test a design at the logic level, a logic level representation of the design is needed. A logic level representation may be derived by submitting the transistor level description of the design to an abstraction mechanism, and having the abstraction mechanism generate an equivalent logic level description. The logic level description may be in a higher level description language such as Verilog HDL. In generating the logic level description, the abstraction mechanism analyzes various combinations of transistors and circuit elements in the transistor level description of the design, and transforms the combinations into elements that perform logic functions (e.g. AND, OR, etc.). By abstracting the functions performed by various transistors and circuit elements, the abstraction mechanism generates a higher level representation of the design, which is simpler and more efficient to simulate. Once derived, the logic level representation may be submitted to an event level simulator for simulation.

[0005] Currently, three basic approaches are used to perform functional abstraction on a transistor level representation. These include pattern matching, transformation rules, and symbolic analysis. With pattern matching, a user specifies to the abstraction mechanism a set of transistor level patterns. The abstraction mechanism then performs the abstraction by looking for all instances of those patterns in the transistor level representation. With transformation rules, the user specifies a set of rules for transforming or replacing portions of a transistor level representation with certain

logic gates. Using these rules, the abstraction mechanism makes the specified transformations throughout the transistor level representation. A major issue with these approaches is that they both require the user to provide a complete set of patterns or transformation rules. As a result, their application is somewhat limited to abstraction of structured designs in which only a limited set of transistor level configurations are used.

[0006] The third approach, symbolic analysis, is an algorithmic technique that abstracts functionality based upon transistor sizes and connectivity. Given a transistor level representation, symbolic analysis generates a functionally equivalent logic level representation using logic primitives and explicit delays (previous state is modeled with an explicit delay). An advantage of symbolic analysis is that it does not require users to specify patterns or transformation rules. Thus, it performs abstraction "automatically". A disadvantage of symbolic analysis is that it outputs a logic level representation that can only be submitted to an event level simulator for simulation. The output of symbolic analysis cannot be submitted to a cycle simulator, or an equivalence checker because it is not a cycle ready model. Symbolic analysis is described in detail in R. E. Bryant, "Extraction of Gate Level Models from Transistor Circuits by Four-Valued Symbolic Analysis," International Conference on Computer Aided Design, 1991, pages 350-353, and in R. E. Bryant, "Boolean Analysis of MOS circuits," IEEE Transactions on Computer Aided Design, July 1987, pages 634-649. Both of these references are incorporated herein in their entirety by reference.

[0007] Overall, the abstraction techniques currently used are relatively rudimentary. They provide some capability for functionally abstracting simple elements, but they do not enable more complex structures, such as precharged latches and flip-flops

to be abstracted (at least not to the latch or flip flop level). Because such structures are quite prevalent in modern component design, such an abstraction capability is needed.

Summary

[0008] In view of the limitations of the prior art, there is provided an abstraction mechanism, which is capable of recognizing and abstracting precharged latches and flip-flops, and which is capable of generating a cycle ready representation of the precharged latches and flip-flops. Because the representation is cycle ready, it can be submitted to a cycle simulator, an equivalence checker, or an emulator for further simulation and emulation.

[0009] According to one embodiment, the abstraction mechanism abstracts precharged latches and flip-flops by using cofactors. In doing so, the abstraction mechanism accesses a logic level representation of a structure. Based upon the logic level representation, the abstraction mechanism derives one or more cofactors. These cofactors are then tested to determine whether they indicate precharge behavior, and whether they indicate latch or flip-flop behavior. If the cofactors indicate both precharge behavior and latch or flip-flop behavior, then the abstraction mechanism abstracts the structure as a precharged latch, or a precharged flip-flop, whichever is appropriate. By recognizing and abstracting precharged latches and flip-flops in this manner, the abstraction mechanism simplifies the functional representation of the structure, and makes it possible to generate a cycle ready representation.

Brief Description of the Drawings

[0010] Fig. 1 is a graphical representation of a transistor level description of a circuit comprising a precharged NAND gate and a latch.

[0011] Fig. 2 shows the circuit of Fig. 1 after it has been separated into channel connected regions and rank ordered.

[0012] Fig. 3 is a logic level representation of the circuit of Fig. 1 generated using symbolic analysis.

[0013] Fig. 4 shows the positive and negative cofactors for the logic level representation of the latch of Fig. 3.

[0014] Fig. 5 shows the logic level representation of Fig. 3 after the latch has been abstracted.

[0015] Fig. 6 shows a cofactor for the logic level representation of the precharged NAND gate of Fig. 3 after the precharge has been propagated and the Clk is restricted to logic 1.

[0016] Fig. 7 shows the circuit of Fig. 1 after the precharged NAND gate and the latch have been abstracted.

[0017] Fig. 8 is a graphical representation of a transistor level description of a sample precharged latch structure.

[0018] Fig. 9 shows the structure of Fig. 8 after it has been rank ordered.

[0019] Fig. 10 is a logic level representation of the structure of Fig. 8 generated using symbolic analysis.

[0020] Fig. 11 shows a first resultant cofactor for the logic level representation of Fig. 8.

[0021] Fig. 12 shows a second resultant cofactor for the logic level representation of Fig. 8.

[0022] Fig. 13 is a flow diagram of one embodiment of a technique for recognizing and abstracting precharged latches.

[0023] Fig. 14 is a graphical representation of a transistor level description of a sample precharged flip-flop structure.

[0024] Fig. 15 shows the structure of Fig. 14 after it has been rank ordered.

[0025] Fig. 16 shows a MUX-XOR equivalent of a logic level representation of the structure of Fig. 14.

[0026] Fig. 17 shows a first resultant cofactor for the MUX-XOR equivalent of Fig. 16.

[0027] Fig. 18 shows a second resultant cofactor for the MUX-XOR equivalent of Fig. 16.

[0028] Fig. 19 shows yet another cofactor for the MUX-XOR equivalent of Fig. 16.

[0029] Fig. 20 shows yet another cofactor for the MUX-XOR equivalent of Fig. 16.

[0030] Fig. 21 is a flow diagram of one embodiment of a technique for recognizing and abstracting precharged flip-flops.

[0031] Fig. 22 is a block diagram of a sample computer system in which one embodiment of the abstraction mechanism may be implemented.

Detailed Description of Embodiment(s)

[0032] Techniques are disclosed below for recognizing and abstracting precharged latches and flip-flops. The techniques are implemented by an abstraction mechanism. In one embodiment, the abstraction mechanism takes the form of a set of computer program instructions executed by one or more processors. In an alternative embodiment, the abstraction mechanism may take the form of customized circuitry. These and other embodiments are within the scope of the present invention.

[0033] Before describing the techniques for recognizing and abstracting precharged latches and flip-flops in detail, some background information will be provided to facilitate a complete understanding of the invention. This background information will be provided with reference to a sample circuit design.

Preparation for Abstraction

[0034] Fig. 1 shows a transistor level representation of a sample circuit design that will be used to illustrate how basic functional abstraction may be carried out. For purposes of explaining the abstraction process, the circuit design is shown in graphical form. In actual implementation, however, the circuit is represented to an abstraction mechanism as a netlist. In the netlist, a description language (such as SPICE or SPECTRE, for example) is used to describe the elements of the circuit and the interconnections between the elements.

[0035] The circuit of Fig. 1 shows a precharged two-input NAND gate 110 followed by a latch structure 120. In the circuit design shown, the latch structure 120 requires proper sizing of the feedback inverter 122. The feedback inverter 122 should be weak so that it can be overdriven by data at node P 130 when Clk 132 is at logic 1. The latch structure 120 will be abstracted as a latch only if the feedback inverter 122 is weak enough to be overdriven by the forward path.

[0036] Analysis of the transistor level representation of the circuit design takes place in stages. In a first stage, an abstraction mechanism analyzes the netlist of the design to distinguish the various channel-connected regions (CCR's) of the circuit. A channel-connected region is a group of all transistors connected by source and drain terminals. Two CCR's are connected by gate terminals. In the circuit of Fig. 1, there are three distinct CCR's. These are shown in Fig. 2.

[0037] After CCR identification, an abstraction mechanism rank orders or levelizes the CCR's from inputs to outputs. During rank ordering, an abstraction mechanism introduces delays to break feedback loops. As shown in Fig. 2, there is a feedback loop between CCR#3 and CCR#2. Thus, a feedback loop delay 210 is introduced to the netlist to break the feedback loop.

[0038] Thereafter, the abstraction mechanism performs symbolic analysis on each of the CCR's to generate an equivalent logic level representation of the circuit. The result of the symbolic analysis process is shown in Fig. 3. As part of the process, charge storage delays such as delay 310 may be introduced to represent the charge storage function of a charge storage element, such as the capacitor 140 of Fig. 1. The process of generating a

logic level representation from a transistor level representation using symbolic analysis is known. Thus, it need not be described in detail herein.

[0039] In the generated logic level representation of Fig. 3, node P 130 is driven by three enable drivers 320, 322, 324. The top enable driver 320 is activated when Clk 132 is at logic 0 and drives node P 130 to logic 1. This driver 320 corresponds to the path from node P 130 to vdd through the PMOS transistor in Fig. 1. The middle enable driver 322 is activated when Clk 132, input A, and input B are all at logic 1, and drives node P 130 to logic 0. This driver 322 corresponds to the path from P 130 to ground through the chain of NMOS transistors in Fig. 1. The bottom enable driver 324 is activated when Clk 132 is at logic 1 and either input A or input B is at logic 0. This driver 324 corresponds to charge storage on node P 130. The delay 310 is used to model the previous value of node P 130 stored on the capacitance 140 (Fig. 1) on node P 130. As noted above, delay 310 is referred to as a charge storage delay.

[0040] A similar explanation can be given for node Out 134. The node Out 134 is driven by two enable drivers 330, 332. The top driver 330 is activated when Clk 132 is at logic 1 and drives node Out 134 to the logic value at node P 130. This driver corresponds to the transparent path from node P 130 to node Out 134 in the netlist of Fig. 1. The bottom enable driver 332 is activated when Clk 132 is at logic 0. When that occurs, node Out 134 stores the previous value of node Out 134. The delay 210 used to model this behavior is the one that was introduced during rank ordering. As noted previously, delay 210 is referred to as a feedback loop delay since it is used to break a feedback loop during rank ordering.

[0041] A functional analysis of the logic level representation shown in Fig. 3 will reveal that the enable drivers 320, 322, 324 on node P 130 capture the functionality of a precharge 2-input NAND gate, and that the enable drivers 330, 332 on Node Out 134 capture the functionality of a latch. Thus, the functionality of the transistor-level representation of Fig. 1 is captured by the logic level representation of Fig. 3. As is, the logic level representation may be submitted to an event-level simulator for simulation. However, because it contains explicit delays, the logic level representation cannot be fed to a cycle simulator, an equivalence checker, or an emulator. In other words, it is not cycle ready. To make the representation of the circuit of Fig. 1 cycle ready, further abstraction is needed to remove the explicit delays, and to abstract latch and flip flop structures.

Recognition and Abstraction of Basic Latch

[0042] As noted above, the logic level representation of Fig. 3 models a precharge two input NAND gate, and a latch. These two structures are recognized and abstracted separately.

[0043] With regard to latch recognition, it is observed that a latch requires an explicit feedback loop to store previous values. Thus, to find a latch candidate, the abstraction mechanism looks for feedback loops. Recall that feedback loop delays are introduced during rank ordering to break up feedback loops; thus, to find a feedback loop, the abstraction mechanism looks for feedback loop delays. According to this observation, the abstraction mechanism finds candidates for a latch output by looking for nodes that provide inputs to feedback loop delays.

[0044] After finding a candidate latch output, the abstraction mechanism tries to identify a latch clock. The abstraction mechanism does this by making a "guess" as to which signal may provide the clock signal for the latch. The intuition for guessing the latch clock is that all paths to a candidate latch output should be controlled by the latch clock. The candidate latch output is driven by a set of enable drivers. Thus, all of those enable drivers should be controlled by the latch clock. Accordingly, the candidate latch clock is identified as the first intersection point in a breadth first traversal of the transitive fanin cones of the control pin of all of the enable drivers on the candidate latch output.

[0045] After a candidate latch clock is identified, the abstraction mechanism performs functional analysis to determine whether the candidate latch output and the candidate latch clock behave as a latch. In doing so, the abstraction mechanism computes both the negative and positive cofactors of the candidate latch output with respect to the candidate latch clock. The negative cofactor is obtained by restricting the candidate latch clock to logic 0 and computing the Boolean equation for the candidate latch output. The positive cofactor is obtained by restricting the candidate latch clock to logic 1 and again computing the Boolean equation for the candidate latch output. For a latch, one of the cofactors should be a feedback loop. The other cofactor should be devoid of a feedback loop. If both of these conditions are met, then a latch is recognized and abstracted, and the cofactor devoid of the feedback loop is identified as the data for the latch.

[0046] To apply this technique, consider the logic level representation shown in Fig. 3. The logic level representation shows node Out 134 as an input to the feedback loop delay 210. Thus, node Out 134 is a candidate for the latch output. There are two enable drivers 330, 332 to node Out 134. The signal Clk 132 is the first intersection point in the

transitive fanin of the control pins of the two enable drivers 330, 332. Thus, Clk 132 is identified as the guess for the latch clock.

[0047] Thereafter, the negative and positive cofactors are computed with respect to Clk 132. The negative cofactor is obtained by restricting Clk 132 to logic 0. With Clk 132 at logic 0, driver 330 is disabled and driver 332 is enabled. Thus, as shown in Fig. 4, the result of restricting Clk 132 to logic 0 is a feedback loop that stores the previous value of node Out 134. The positive cofactor is obtained by restricting Clk 132 to logic 1. When this is done, driver 332 is disabled and driver 330 is enabled. The result, as shown in Fig. 4, is that node Out 134 takes on the data value on node P 130. Since one of the cofactors (the negative cofactor) is a feedback loop, and the other cofactor (the positive cofactor) is devoid of a feedback loop, both latch conditions are satisfied. Thus, a latch is recognized and abstracted with the positive cofactor identified as the latch data. This result is shown in Fig. 5.

Recognition and Abstraction of Basic Precharge

[0048] A different technique is used to recognize the precharge on the precharge NAND gate. With precharge recognition, a user specifies a clocking scheme. In the following example, it will be assumed for the sake of simplicity that the clocking scheme is a two-phase scheme. However, it should be noted that other clocking schemes (such as a 4 phase non-overlapping scheme) may be specified.

[0049] It is observed that precharge structures need capacitive nodes to store the previous value of a node during the evaluate phase. Since capacitive nodes are modeled using charge storage delays, charge storage delays indicate candidates for precharge

recognition. Specifically, the node that provides the input to the charge storage delay becomes a precharge candidate.

[0050] After a precharge candidate is identified, the abstraction mechanism performs cofactor computations on the logic level representation with respect to each phase of the user-specified clocking scheme to determine whether precharge behavior is exhibited. If a precharge candidate precharges to logic 1 during one of the clock phases, then it is marked as an evaluate candidate for the subsequent clock phases. Cofactor computations are then performed for the candidate evaluate node under the assumption that the previous value of the candidate evaluate node is logic 1. After the cofactor computations are performed, a set of symbolic manipulations and optimizations are performed on the cofactor during the evaluate phase. The resultant cofactor represents the results of precharge recognition on the precharge candidate.

[0051] To apply this technique, consider the logic level representation shown in Figure 5, which is obtained after latch recognition and abstraction has been performed. As shown, the charge storage delay 310 receives its input from node P 130. Thus, node P 130 becomes a precharge candidate. Assume that the user specifies a simple two phase clocking scheme, where Clk 132 is logic 0 in the first phase and logic 1 in the second phase. Cofactor computation is performed on node P 130 by initially restricting Clk 132 to logic 0 (the first phase of the Clk 132). With Clk 132 at logic 0, drivers 322 and 324 are disabled, and driver 320 is enabled; thus, node P 130 precharges to logic 1. As a result, node P 130 is identified as a candidate evaluate node. Since Clk 132 is at logic 0, the precharge is not consumed by the downstream latch.

[0052] Cofactor computation is then performed on the candidate evaluate node P 132 under the assumption that the previous value of the node is a logic 1. To do this, the precharge on node P 130 is propagated through the charge storage delay 310. This operation can be viewed as a "stuck at logic 1" condition on node PrevP 510. The "stuck at logic 1" condition models the fact that the previous value of node P 130 is logic 1. Cofactor computation is then performed on the logic level representation for the second phase of the Clk 132 (i.e. Clk 132 is restricted to logic 1). The resultant cofactor for this computation is shown in Fig. 6. Functional analysis of this logic level representation reveals that the drivers 322, 324 and inverter 610 behave as an inverter. Thus, applying a set of symbolic manipulations and optimizations to the gate lever representation produces the equivalent NAND representation shown in Fig. 7. The precharge NAND gate is thus recognized and abstracted. This abstraction of the precharge NAND gate is valid under the assumption that the inputs (A and B) are held stable during the evaluate phase of the Clk 132.

Recognition and Abstraction of Precharged Latches

[0053] The techniques discussed above may be used to recognize and abstract basic latches and precharge on basic gates. They cannot, however, be used to recognize and abstract a precharged latch. For such abstraction, a significantly revised technique is needed.

[0054] To describe one embodiment of a technique that may be used to recognize and abstract a precharged latch, reference will be made to the sample circuit of Fig. 8. Fig. 8 shows a transistor level representation of a precharged latch. Reference will also

be made to Fig. 13, which shows a flow diagram for one embodiment of the technique. The operations shown in the flow diagram are performed by an abstraction mechanism.

[0055] As shown in Fig. 13, the abstraction mechanism begins analysis of the precharged latch structure of Fig. 8 by preparing (1304) the structure for abstraction. In one embodiment, the structure is submitted to the abstraction mechanism in the form of a netlist. The abstraction mechanism performs a number of actions on this netlist to prepare it for functional abstraction. In one embodiment, the abstraction mechanism initially analyzes the netlist to determine all of the different CCR's in the netlist. The abstraction mechanism then rank orders the CCR's from inputs to outputs. During rank ordering, the abstraction mechanism introduces delays to break feedback loops. In the structure of Fig. 8, there is a feedback loop at node Out 810. Thus, a feedback loop delay 910 (Fig. 9) is introduced into the netlist to break the feedback loop. The result of rank ordering is shown in Fig. 9. After the netlist is rank ordered, the abstraction mechanism performs symbolic analysis on the netlist to generate a logic level representation of the structure. The result of symbolic analysis is shown in Fig. 10.

[0056] As shown, node P 820 is driven by three enable drivers 1010, 1012, 1014. The top enable driver 1010 precharges node P 820 when Clk 830 is at logic 0. The middle enable driver 1012 is activated when both Clk 830 and d 840 (the data signal) are at logic 1. The bottom enable driver 1014 is activated when Clk 830 is at logic 1 and d 840 is at logic 0. This driver 1014 corresponds to charge storage on node P 820 (notice the charge storage delay 1020 introduced during symbolic analysis). The node Out 810 is driven by a pair of cross-coupled NAND gates. The feedback loop between the cross-

coupled NAND gates is broken by the feedback loop delay 910 introduced during rank ordering.

[0057] After the initial logic level representation shown in Fig. 10 is generated, the abstraction mechanism proceeds to attempt to further abstract the structure. In doing so, the abstraction mechanism accesses (1308) the logic level representation of Fig. 10. Based upon the logic level representation, the abstraction mechanism derives (1312) a first resultant cofactor. In one embodiment, the abstraction mechanism derives the cofactor by performing cofactor computation on the logic level representation while restricting the Clk 830 to a certain logic value.

[0058] To illustrate, assume that a user specifies a simple two phase clocking scheme, where Clk 830 is at logic 0 on the first phase and logic 1 on the second phase (note: other clocking schemes may be specified if so desired). Assume further that the first resultant cofactor is computed on the first phase of the Clk 830 (i.e. Clk 830 is restricted to logic 0). With Clk 830 restricted to 0, the resultant cofactor for the logic level representation is that shown in Fig. 11. This cofactor shows that when Clk 830 is at logic 0, node P 820 precharges to logic 1, and node Out 810 holds its previous value.

[0059] After the first resultant cofactor is computed, the abstraction mechanism proceeds to determine (1316) whether the cofactor indicates: (1) that node P 820 is experiencing a precharge; and (2) that node Out 810 is part of a feedback loop. If the cofactor indicates both to be true, then in one embodiment, the abstraction mechanism concludes that node Out 810 is a precharged latch candidate. Otherwise, the abstraction mechanism may conclude (1332) that the structure is not a precharged latch.

[0060] In the present example, the first resultant cofactor indicates that node Out 810 is a precharged latch candidate. Thus, the abstraction mechanism proceeds to derive (1320) a second cofactor. In one embodiment, the second cofactor is derived by propagating the precharge through a charge storage delay 1020, and then performing cofactor computations on the logic level representation based upon another phase of the Clk 830.

[0061] With reference to the present example, the second resultant cofactor is derived by first propagating the precharge on node P 820 through the charge storage delay 1020. This operation can be viewed as a "stuck at 1" condition at node PrevP 1030. The "stuck at 1" condition models the fact that the previous value of node P 820 was logic 1. Hence, the current value of node PrevP 1030 is now logic 1. With node PrevP 1030 set at logic 1, the abstraction mechanism performs cofactor computation on the logic level representation based upon a second phase of the Clk 830 (i.e. Clk 830 is restricted to logic 1). The resultant cofactor with the "stuck at 1" assumption is shown in Fig. 12. With some symbolic manipulations and optimizations, this cofactor reveals that there is a transparent path from the data signal d 840 to node Out 810.

[0062] After the second resultant cofactor is derived, the abstraction mechanism determines (1324) whether the second resultant cofactor is devoid of a feedback loop. If so, then it is determined that all of the indications of a precharged latch are found. Namely, the cofactors indicate: (1) that a precharge is experienced; (2) that one of the cofactors indicates a feedback loop is present to hold a previous value; and (3) that the other cofactor is devoid of a feedback loop. If that is the case, the structure represented by the logic level representation is recognized and abstracted (1328) as a precharged

latch. On the other hand, if the second cofactor is not devoid of a feedback loop, then the abstraction mechanism may conclude (1332) that the structure is not a precharged latch. In the present example, the second resultant cofactor is devoid of a feedback loop. Consequently, the structure of Fig. 10 is recognized and abstracted as a precharged latch with the second cofactor identified as the data for the latch.

Recognition and Abstraction of Precharged Flip-flops

[0063] A revised technique is used to recognize and abstract the more complex structure of a precharged flip-flop. To describe this technique, reference will be made to the sample circuit shown in Fig. 14, which depicts a graphical representation of a transistor level netlist for a precharged flip-flop. This flip-flop uses a precharge phase to precharge nodes P1 1410 and P2 1412 to logic 1. The cross-coupled NAND gates 1450, 1452 (for the sake of simplicity, the NAND gates are shown as gates rather than their transistor level equivalents) at node Out 1420 hold the previous state during the precharge phase. At the positive edge of Clk 1430, node P1 1410 gets the inverted form of the logic value at data node (or data signal) d 1440. Node P2 1412 gets the logic value at data node d 1440. Node Out 1420 gets the logic value at node d 1440. Note that this circuit functions as a flip-flop since the logic value at node d 1440 propagates to the output node Out 1420 only at the positive edge of Clk 1430.

[0064] To describe one embodiment of a technique that may be used to recognize and abstract a precharged flip-flop, reference will be made to the flow diagram of Fig. 21. In one embodiment, the technique of Fig. 21 is implemented by an abstraction mechanism. As shown, the abstraction mechanism begins analysis of the precharged flip-flop

structure by preparing (2104) the structure for abstraction. In doing so, the abstraction mechanism analyzes the netlist to determine all of the different CCR's in the netlist. The abstraction mechanism then rank orders the CCR's from inputs to outputs. During rank ordering, the abstraction mechanism introduces delays to break feedback loops. In the structure of Fig. 14, there are three feedback loops. Thus, three delays are introduced into the netlist. The result of rank ordering is shown in Fig. 15, with the three delays 1510, 1512, and 1514 clearly indicated.

[0065] After the netlist is rank ordered, the abstraction mechanism performs symbolic analysis on the netlist to generate a logic level representation of the structure. In this case, symbolic analysis generates a rather complicated logic level representation in terms of AND, NOT, and DELAY primitives. Rather than showing this complex logic level representation, a MUX-XOR equivalent of the logic level representation is shown in Fig. 16 to facilitate discussion. The MUX-XOR equivalent captures all of the functionality of the generated logic level representation. According to the MUX-XOR equivalent, when the Clk 1430 is at logic 0, nodes P1 1410 and P2 1412 precharge to logic 1. The cross-coupled NAND gates driving node Out 1420 will hold the previous state. When Clk 1430 goes to logic 1, values at nodes P1 1410 and P2 1412 will depend on the XOR of the previous values on nodes P1 1410 and P2 1412. At the positive edge of Clk 1430, the previous values on nodes P1 1410and P2 1412 will be logic 1 (the precharged values). Thus, the XOR computes to logic 0 and node P1 1410 gets the inverted form of the value at node d 1440, and node P2 1412 gets the value at node d 1440. Consequently, node Out 1420 obtains the value at node d 1440. Now, nodes P1 1410 and P2 1412 are complements of each other. As a result, the XOR evaluates to logic 1, and nodes P1 1410

and P2 1412 hold their previous values. This implies that node Out 1420 is insulated from changes on node d 1440 when Clk 1430 is at logic 1. As this discussion shows, the MUX-XOR equivalent correctly captures the functionality of a flip-flop.

[0066] Returning to Fig. 21, after the structure is prepared for abstraction, the abstraction mechanism proceeds to attempt to further abstract the structure. In doing so, the abstraction mechanism accesses (2108) the logic level representation of the structure. Based upon the logic level representation, the abstraction mechanism derives (2112) a first resultant cofactor. In one embodiment, the abstraction mechanism derives the cofactor by performing cofactor computation on the logic level representation while restricting the Clk 1430 to a certain logic value.

[0067] To illustrate, assume that a user specifies a simple two phase clocking scheme, where Clk 1430 is at logic 0 on the first phase and logic 1 on the second phase (note: other clocking schemes may be specified if so desired). Assume further that the first resultant cofactor is computed on the first phase of the Clk 1430 (i.e. Clk 1430 is restricted to logic 0). With Clk 1430 at logic 0, the resultant cofactor for the logic level representation is that shown in Fig. 17. As shown, nodes P1 1410 and P2 1420 precharge to logic 1. Node Out 1420 holds its previous value.

[0068] After the first resultant cofactor is computed, the abstraction mechanism proceeds to determine (2116) whether the cofactor indicates: (1) that node P1 1410 is experiencing a precharge; (2) that node P2 1412 is experiencing a precharge; and (3) that node Out 1420 is part of a feedback loop. If the cofactor indicates that all of these are true, then the abstraction mechanism concludes that node Out 1420 is a precharged flip-

flop candidate. Otherwise, the abstraction mechanism may conclude (2136) that the structure is not a precharged flip-flop.

[0069] In the present example, the first resultant cofactor indicates that node Out 1420 is a precharged flip-flop candidate. Thus, the abstraction mechanism proceeds to derive (2120) a second resultant cofactor. In one embodiment, the second cofactor is derived by propagating the precharges through the delays 1512, 1514, and then performing cofactor computation on the logic level representation based upon another phase of the Clk 1420.

[0070] With reference to the present example, the second resultant cofactor is derived by propagating the precharge on node P1 1410 through delay 1512, and the precharge on node P2 1412 through delay 1514. This operation can be viewed as a "stuck at 1" condition on nodes PrevP1 1610 and PrevP2 1612. The "stuck at 1" condition models the fact that the previous values on nodes P1 1410 and P2 1412 were logic 1. Hence, the current values on nodes PrevP1 1610 and PrevP2 1612 are now logic 1.

[0071] With nodes PrevP1 1610 and PrevP2 1612 set at logic 1, the abstraction mechanism performs cofactor computation on the logic level representation based upon a second phase of the Clk 1430 (i.e. Clk 1430 is restricted to logic 1). The resultant cofactor with the "stuck at 1" assumption is shown in Fig. 18. With some symbolic manipulations and optimizations, this cofactor can be optimized as shown in Fig. 18 to reveal that there is a transparent path from the data node d 1440 to node Out 1420.

[0072] After the second resultant cofactor is derived, the abstraction mechanism determines (2124) whether the second cofactor is devoid of a feedback loop. If so, then the precharged flip-flop candidate is still a candidate. If not, then the abstraction

mechanism may conclude (2136) that the structure is not a precharged flip-flop. In the present example, the second cofactor is devoid of a feedback loop. Thus, the abstraction mechanism proceeds further with the abstraction process.

[0073] In proceeding further, the abstraction mechanism determines (2128) whether, under certain conditions, the precharged flip-flop candidate (node Out 1420) takes on logic values that are independent of the data on data node d 1440. If so, then the structure can be abstracted (2132) as a precharged flip-flop. If not, then the abstraction mechanism may conclude (2136) that the structure is not a precharged flip-flop. In one embodiment, the abstraction mechanism makes the data independence determination by further computing cofactors.

[0074] Specifically, it is recognized that the value at node Out 1420 when Clk 1430 is at logic 1 is a function of the previous logic values on nodes P1 1410 and P2 1412. The second resultant cofactor derived above with Clk 1430 at logic 1 was done under the "stuck at 1" assumption since logic 1 was the previous value on nodes P1 1410 and P2 1412 as a result of the precharge. It is observed, however, that after the positive edge of Clk 1430, nodes P1 1410 and P2 1412 are no longer both at logic 1 but rather become complements of each other. That being the case, the abstraction mechanism computes two additional cofactors with Clk 1430 at logic 1, assuming that the previous values of nodes P1 1410 and P2 1412 are complements of each other. This corresponds to a cofactor computation with Clk at logic 1, node PrevP1 1610 at logic 1, and node PrevP2 1612 at logic 0, and another cofactor computation with Clk 1430 at logic 1, node PrevP1 1610 at logic 0, and node PrevP2 1612 at logic 1. The cofactor for the case with Clk at logic 1, node PrevP1 1610 at logic 1, and node PrevP2 1612 at logic 0 is shown in Fig.

19. The cofactor for the case with Clk at logic 1, node PrevP1 1610 at logic 0, and node PrevP2 1612 at logic 1 is shown in Fig. 20. In both cases, the logic value at node Out 1420 evaluates to a constant value, which means that the logic value at node Out 1420 is independent of the logic value on the data node d 1440. This implies that once nodes P1 1410 and P2 1412 become complements of each other, node Out 1420 is insulated from changes at node d 1440. This indicates flip-flop behavior. Thus, the structure can be abstracted as a precharged flip-flop.

Hardware Overview

[0075] In one embodiment, the abstraction mechanism of the present invention is implemented as a set of instructions executable by one or more processors. The invention may be implemented, for example, as part of an object oriented programming system. Fig. 22 shows a hardware block diagram of a computer system 2200 on which the abstraction mechanism may be implemented. Computer system 2200 includes a bus 2202 or other communication mechanism for communicating information, and a processor 2204 coupled with bus 2202 for processing information. Computer system 2200 also includes a main memory 2206, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 2202 for storing information and instructions to be executed by processor 2204. Main memory 2206 may also be further used to store temporary variables or other intermediate information during execution of instructions by processor 2204. Computer system 2200 further includes a read only memory (ROM) 2208 or other static storage device coupled to bus 2202 for storing static information and instructions for processor 2204. A storage device 2210, such as a

magnetic disk or optical disk, is provided and coupled to bus 2202 for storing information and instructions.

[0076] Computer system 2200 may be coupled via bus 2202 to a display 2212, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 2214, including alphanumeric and other keys, is coupled to bus 2202 for communicating information and command selections to processor 2204. Another type of user input device is cursor control 2216, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 2204 and for controlling cursor movement on display 2212. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0077] According to one embodiment, the functionality of the present invention is provided by computer system 2200 in response to processor 2204 executing one or more sequences of one or more instructions contained in main memory 2206. Such instructions may be read into main memory 2206 from another computer-readable medium, such as storage device 2210. Execution of the sequences of instructions contained in main memory 2206 causes processor 2204 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0078] The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 2204 for execution. Such a

medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 2210. Volatile media includes dynamic memory, such as main memory 2206. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 2202. Transmission media can also take the form of acoustic or electromagnetic waves, such as those generated during radio-wave, infra-red, and optical data communications.

[0079] Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0080] Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 2204 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 2200 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 2202. Bus 2202 carries the data to main memory 2206, from which processor 2204 retrieves and executes

the instructions. The instructions received by main memory 2206 may optionally be stored on storage device 2210 either before or after execution by processor 2204.

[0081] Computer system 2200 also includes a communication interface 2218 coupled to bus 2202. Communication interface 2218 provides a two-way data communication coupling to a network link 2220 that is connected to a local network 2222. For example, communication interface 2218 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 2218 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 2218 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0082] Network link 2220 typically provides data communication through one or more networks to other data devices. For example, network link 2220 may provide a connection through local network 2222 to a host computer 2224 or to data equipment operated by an Internet Service Provider (ISP) 2226. ISP 2226 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 2228. Local network 2222 and Internet 2228 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 2220 and through communication interface 2218, which carry the digital data to and from computer system 2200, are exemplary forms of carrier waves transporting the information.

[0083] Computer system 2200 can send messages and receive data, including program code, through the network(s), network link 2220 and communication interface 2218. In the Internet example, a server 2230 might transmit a requested code for an application program through Internet 2228, ISP 2226, local network 2222 and communication interface 2218. The received code may be executed by processor 2204 as it is received, and/or stored in storage device 2210, or other non-volatile storage for later execution. In this manner, computer system 2200 may obtain application code in the form of a carrier wave.

[0084] At this point, it should be noted that although the invention has been described with reference to a specific embodiment, it should not be construed to be so limited. Various modifications may be made by those of ordinary skill in the art with the benefit of this disclosure without departing from the spirit of the invention. Thus, the invention should not be limited by the specific embodiments used to illustrate it but only by the scope of the appended claims.